HW 4 – Documentation

Itamar Barron – 208981159
Roy Frumkis – 312472129

Introduction

The purpose of this project is to enhance the xv6 operating system by changing the round-robin scheduler in xv6 way of dealing with tasks by adding them priority.
The current scheduler assigns equal priority to all processes, but our objective is to introduce user-defined process priorities and incorporate them into the scheduler.
Our task involves implementing a priority, denoted as *priority*, for each process in range from 0 (lowest) to 7 (highest). By default, all processes start with priority 0, but we will introduce new system calls to enable users to set and retrieve process priorities.
The setprio(priority) system call will allow a process to specify its desired priority, while the getprio() system call will provide a means to verify the priority set for a process.

The **setprio(priority)** system call takes an integer value *priority* as an argument and is used to set the priority of the calling process. The function updates the priority value associated with the process to the specified value, allowing dynamic adjustment of process priorities.

The **getprio()** system call is used to retrieve the current priority value of the calling process. It takes no arguments and returns the priority value associated with the process.

To incorporate the priority-based scheduling, we have modified the existing xv6 scheduler. The priority value *priority* is determining the number of consecutive timers ticks a process can receive during the round-robin scanning of processes. As long as a process remains runnable (not sleeping), it will be scheduled 2^P times in sequence before the scheduler considers the next process. However, if a process needs to sleep and becomes non-runnable before completing its 2^P ticks, it will not be finished, and the scheduler will proceed to the next process.

To demonstrate the functionality of the modified scheduler, we have included a user-space program called “demosched.c”. This program spawns 8 child processes that execute “some_work” code with different priorities. Through this demonstration, we aim to showcase that processes finish their computations in accordance with the set priorities.
This documentation provides a comprehensive guide to understanding the modifications made to the xv6 scheduler.

Demonsched.c user space program

The demosched.c program has been created to showcase the enhanced scheduler with priority support in the xv6 operating system. Its purpose is to demonstrate how processes can now be assigned different priorities and how the scheduler handles their execution accordingly.

The program starts by forking 8 child processes, each representing a computational task. Using the setprio system call, different priorities are assigned to these child processes and once the priorities are set, the child processes execute a computational task, emulating resource-intensive tasks that utilize significant processing power. The task is calculating the sum of getprio() * getprio() + getprio() for 50000 iterations. Demonstrating the get_prio() functionality too.

The parent process, or the main function, keeps track of the child processes' execution by utilizing the wait() system call. As each child process completes its task, the parent process records the execution time. By comparing the execution times of the child processes with different priorities, it becomes apparent whether the scheduler functions according to the set priorities. If the processes finish computation in accordance with their assigned priorities, it provides validation that the modified scheduler is appropriately considering the priorities during resource allocation.

Other changes

To create this functionality, we have made changes in the origin xv6 files which we will summarize and explain; for every file we will detail the changes and elaborate on each.

Makefile -
In order to create new functionalities, we have change UPROGS (by adding _demosched) and EXTRA (by adding demosched.c) so when calling the new system call the OS will be familiar with it.

Proc.c -
In the proc.c file we did the following changes

- Added implementation for the **setprio()** function. This function takes an integer argument representing the desired priority level and sets the priority of the calling process accordingly. It updates the priority field associated with the process. The implementation involves modifying the data structure of the process and adjusting the priority value.
- The **scheduler** function has been modified to implement a priority-based scheduling. When a process is chosen, it is run for 2^priority ticks, unless it's status has changed from runnable, meaning it should not be run for the next iteration.
- The **allocproc** function has been extended to include the assignment of a default priority of 0 to the newly allocated process.

Proc.h -
Here, we only had to modify the proc struct and added priority variable that indicates the priority of the process (varying from 0-7).

syscall.c -
In this section we have added the externs and their SYS calls.
Extern int for the two functions we have asked to create and their SYS calls respectevely.

syscall.h -
In the syscall header file we have added new defines of the two asked functions and their right follow's number.

Sysproc.c -
Added implementation for the following two new system call functions:
int sys_setprio(int Priority): Defines a system call that invokes the setprio() function from proc.c to set the priority of the calling process. It takes one argument, int priority, specifying the desired priority level.
int sys_getprio(void): Defines a system call that calls that utilizes the myproc function to get current process priority.

User.h -
We have declaired on the two new functions we have created:
getprioc(void), setprio(int priority).

usys.S -
In this file we have declaired out new SYS calls. Here we added the SYSCALL(func) two new lines and this is what makes the functions callable.

Defs.h-
Here we added the setprio(int) declaration to have the function accessed from multiple files.

For all in all, these are all the changes we have made in the origin files in order to make our code completely work. Of course we have wrapped this whole folder as tgz file and set everything to be aligned with the submission guidelines.